REMARKS

The first element of the claim at issue is "duplicating the member data for said class for each application in a shared memory." It is respectfully submitted that this language has been misinterpreted to cover "the applications execute on shared memory." See the bottom of page 5 of the final rejection.

The theory is that because the cited reference teaches a virtual machine, it teaches executing applications on a shared memory because the applications are executed on the virtual machine which is stored in a memory. But, even if it is assumed that the virtual machine is in one memory, that assumption is not commensurate with the scope of the claim. The claim calls for duplicating the member data for said class for each of application in a shared memory. The virtual machine can, necessarily, access whatever memories it wants, be it shared or otherwise. Those memories need not be part of the virtual machine, but may be accessed by it. Thus, the virtual machine is running on a processor, but it is not executing in memory itself. It must access data from memories and the claim requires that those memories be shared. Even if a memory stores the virtual machine instructions that are executed by some type of processor together in some memory, this does not mean that all the data processed by that processor is also stored in the same memory.

For example, the software that makes up the virtual machine may be executed by a processor that accesses a memory that stores that virtual machine software. That same processor then, in the course of executing those instructions, may access data from other memories. Those memories do not need to be shared memories. There is no reason why the member data for each class must be duplicated in a shared memory. It can be provided in non-shared memories and accessed by the processor that is executing the virtual machine.

Thus, whether or not the virtual machine itself is stored in common memory, there is no reason to believe that the member data would be stored in necessarily that same memory or in shared memory.

There is also a citation to the Microsoft Computer Dictionary definition of "shared memory." That definition defines "shared memory" as memory accessed by more than one program in a multitasking environment or a portion of memory used by parallel-processor computer systems to exchange information. Thus, "shared memory" has nothing to do with

where the virtual machine is stored and what kind of memory the virtual machine is stored on. All that matters in the claim is that the member data duplicated in a shared memory which then may be accessed by the processor that is executing the virtual machine. There is absolutely no reason to believe that there is any kind of shared memory for said class for each application in the shared memory.

Therefore, for this first reason, the rejection fails to make out a *prima facie* rejection.

The second issue in the claim at this point is "providing a handle to each application to enable each application to access its member data in shared memory." Since the reference fails to teach duplicating the member data for said class for each application in shared memory, it cannot teach enabling each application to access its member data in the shared memory.

Initially, the Examiner suggests that the Chappell reference teaches, at pages 58-59, that "when a client has an object created, that a reference is returned to the client of the created object." However, nothing in the cited material appears to support that argument.

Further, it is suggested that the Meyer book teaches a client calling a create function that creates an object and associates it with a reference. The claim calls for providing a handle to each application to enable each application to access its member data. It has nothing to do with calling a create function that creates an object and associates it with a reference.

The Examiner's original argument was that the assignment of the reference happens automatically, but the book makes it clear that this is not true. The book states that "The general rule is that, unless you do something to it, a reference remains void." Further, the book explains "To change this, you must create a new object of the appropriate class type and associate it with the reference." In other words, the assertion of automatic operation is plainly refuted by the cited book. For example, the assertion in the final rejection of well known art, on pages 2 and 3, was that "it is also well known in the art that when a client instantiates another class, i.e. access methods class, a reference to that object is returned to the requesting object and, therefore, the applications would receive a reference to the method class when it is instantiated for accessing the application's respective member data."

In view of the Examiner's citation, it is clear that this is an untrue assertion of well known art. Whether the reference is returned depends on whether the new object is created so as to be associated with the reference. Otherwise, the references remain void.

Therefore, a *prima facie* rejection is not made out.

With respect to the assertion that the claims allow for indirect address accessing a member data, the Applicant is at a loss to see where such an interpretation is made, any basis for it, or what, in fact, the Examiner even means by such an assertion. Suffice it to say that no rejection to date has been premised on such a theory. If such a theory is advanced, the final rejection should be withdrawn.

Therefore, reconsideration is requested.

Respectfully submitted,

Date: April 26, 2005

Timothy N. Trop, Reg. No. 28,994
TROP, PRUNER & HU, P.C.
8554 Katy Freeway, Ste. 100
Houston, TX 77024
713/468-8880 [Phone]
713/468-8883 [Fax]

Attorneys for Intel Corporation